10/755,020 PTO-892

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
26 April 2001 (26.04.2001)

PCT

(10) International Publication Number
WO 01/29752 A2

(51) International Patent Classification[7]: G06K

(21) International Application Number: PCT/US00/29048

(22) International Filing Date: 19 October 2000 (19.10.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/420,667 19 October 1999 (19.10.1999) US

(71) Applicant: PULSAR DNA, INC. [US/US]; Suite 5A, 20 East 53rd Street, New York, NY 10022 (US).

(72) Inventor: CRISCIONE, Enzo; Via Jesi, 27, I-60100 Ancona (IT).

(74) Agent: BOVENKAMP, Christopher, T.; Hughes & Luce, L.L.P., Suite 2800, 1717 Main Street, Dallas, TX 75210 (US).

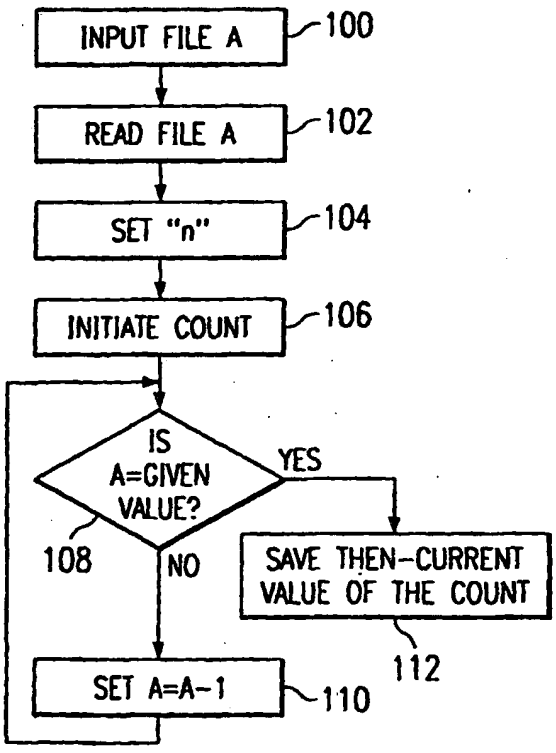
(81) Designated States *(national)*: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW.

(84) Designated States *(regional)*: European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published:
— *Without international search report and to be republished upon receipt of that report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: DATA FILE COMPRESSION AND DECOMPRESSION METHOD



(57) Abstract: Very large compression ratios are obtained by compressing a digital file in a temporal or "time-based" manner. In a preferred embodiment, a compressed version of the file is a representation of the amount of time required to process the digital file against a given function. As the digital file is processed against the given function, a count (e.g., a timer) is incremented at a given rate. The size of the file determines how long the file will take to be processed by the given function. When the processing is complete, a value of the timer is then converted into a representation of the file. The digital file is later reconstructed from the representation by applying an inverse of the given function.

WO 01/29752 A2

DATA FILE COMPRESSION AND DECOMPRESSION METHOD

Technical Field

The present invention relates generally to data conversion and, in particular, to lossless data compression and decompression.

Description of the Related Art

Data compression is a well-defined art. Thus, for example, lossless compression, as the name implies, denotes a compression scheme in which the decompressed or reconstructed data exactly matches the original file. Lossless compression techniques include such techniques as arithmetic coding, entropy coding, Huffman coding, predictive coding, Universal coding, and Ziv-Lempel coding. Arithmetic coding is a technique that maps source sequences to intervals between a pair of numbers. Entropy coding involves fixed-to-variable length coding of statistically independent source symbols. Huffman coding schemes map source symbols to binary codewords of integral length such that no codeword is a prefix of a longer codeword. Predictive coding is a form of coding where a prediction is made for the current event based on previous events and the error in prediction is transmitted. The JPEG standard, which finds widespread use on the Internet, is based on a linear predictive coding technique. Universal coding is a form of coding that is designed without knowledge of source statistics but that converges to an optimal code as the source sequence length increases. Ziv-Lempel coding is a family of dictionary coding-based schemes that encode strings of symbols by sending information about their location in a dictionary. Thus, for example, the LZ77 family uses a portion of the already encoded string as a dictionary, and the LZ78 family builds a dictionary of strings that are encountered in the data stream. The LZ77-based schemes are used commercially to create zip files, while the LZ78-based schemes find widespread use, for example, in GIF compression of image data.

While such techniques are well-known in the art, their performance, when measured in terms of compression ratio, is surprisingly modest. A compression ratio of a given compression scheme is defined as the ratio of the size of the original data versus the size of the compression data. Current data compression algorithms are capable of up to 10:1 lossless compression in true-fidelity scenarios,

in which none of the original data may be sacrificed during the reduction is size; 25:1 lossy compression in high-fidelity scenarios (largely audio, video, and still photography), in which some of the original data may be omitted while still preserving the operative sensory elements of the data; and up to 200:1 lossless compression of sparse files, i.e., files with extremely large amounts of repetitive, unnecessary, or blank data, such as facsimile transmissions.

There remains a long-felt need in the art to provide new compression and decompression schemes that provide significantly higher compression ratios as compared to the known prior art.

The present invention solves this problem.

BRIEF SUMMARY OF THE INVENTION

It is a primary object of the present invention to provide a lossless data compression scheme with millicompression (1000:1) or higher compression ratios, irrespective of the size of the source file.

It is another primary object of this invention is to provide a novel compression/decompression technique that has widespread software and hardware applications.

A further primary object of this invention is to provide a lossless data compression coder-decoder (a "codec") that may implemented in software or hardware and that may be used to facilitate very large scale data compression for numerous applications including, among others, telecommunications, aerospace, commercial, industrial, and consumer applications.

These and other objects of the present invention are achieved by processing a digital file to be compressed in a time-based or "temporal" manner. In one preferred embodiment, the compressed version of the digital file is a representation of the amount of time required to process the digital file against a given mathematical function. As the digital file is processed against the given function, a count (e.g., a timer) is incremented at a given rate. The size of the file determines how long the file will take to be processed by the given function. When the processing is complete, a value of the timer is then converted into a representation of the file. The digital file is later reconstructed from the representation by applying an inverse of the given function.

In a preferred embodiment, the given function includes the following iterative steps. A bit string comprising the file is decremented against a given value (e.g., 1). After decrementing the bit string against the given value, a test is made to determine whether the given value equals an end value (e.g., 0). If not, the decrementing step is repeated against the result of the first iteration. The result is then tested to determine whether the end value has been reached. This process is then repeated for however long is required to reach the end value. The time is then encoded as a representation of the file.

The foregoing has outlined some of the more pertinent objects and features of the present invention. These objects should be construed to be merely illustrative of some of the more prominent features and applications of the invention. Many other beneficial results can be attained by applying the disclosed invention in a different manner or modifying the invention as will be described. Accordingly, other objects and a fuller understanding of the invention may be had by referring to the following Detailed Description of the Preferred Embodiment.

BRIEF DESCRIPTION OF THE DRAWINGS

For a more complete understanding of the present invention and the advantages thereof, reference should be made to the following Detailed Description taken in connection with the accompanying drawings in which:

Figure 1 is a flowchart of a preferred routine for compressing a file according to the present invention;

Figure 2 is a flowchart of a preferred routine for decompressing the file that has been compressed according to the routine of **Figure 1**;

Figure 3 is a flowchart of a more detailed embodiment for compressing a file according to the present invention;

Figure 4 is a flowchart of a more detailed embodiment for decompressing the file that has been compressed according to the routine of **Figure 3**; and

Figure 5 illustrates a preferred chip-based implementation of the inventive routines.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Figure 1 illustrates the basic operation of the inventive compression method. Figure 2 illustrates a corresponding decompression routine. The compression routine begins at step 100 to input the file A to be compressed. Regardless of its size, the file comprises a bit string. The file may be of any given format, e.g., sound, video, image, data, or the like. The file may also represent a compressed form of a file, e.g., a .zip file. At step 102, the file to be compressed is read. At step 104, an index n is set. For illustrative purposes, the index n = 1. At step 106, a count is initiated. A convenient way to increment the count is to initiate a timer from zero and count given time increments at a given rate. A test is then performed at step 108 to determine whether the file, as represented by the bit string, is equal to a given end value. For illustrative purposes, the given value is zero, although this is not a limitation of the present invention. If the output of the test at step 108 indicates that the file is not equal to the given end value, the routine continues at step 110 to apply a given function to the file. In the illustrative example, the given function decrements the file, namely, the bit string, by the index n. In this example, the given function sets A = A – 1. Control then returns back to step 108. When the outcome of the test at step 108 indicates that the file has reached the given end file, the routine branches to step 112 to save the then-current value of the count. The then-current value of the count represents the compressed version of the file. If desired, the then-current value of the count may be augmented with additional information, e.g., a file name, a timestamp, an identifier of the processor used to execute the routine, the processor's clock speed, the count rate, or the like).

Figure 2 is a flowchart illustrating a preferred routine for decompressing the file that is compressed according to the routine of Figure 1. To decompress the file (and thus reconstruct the file A), the routine begins at step 200 by reading the then-current value of the count that was saved at step 112. At step 202, the index "n" is set to the sam e value that was set at step 104 during the compression routine. In the illustrative example, step 202 sets n = 1. At step 204, a variable "count" is set equal to "-count" and then increm ented at the same given rate that was used during the compression routine. Thus, in the example, the

count is a timer that begins at a certain negative value and increments, at the given rate, to a final value. At step 206, a test is preformed to determine whether the count has reached the final value. If not, the routine continues at step 208 to set A equal to a given function, in this case, A = A + 1. Control then returns to step 206 to test whether the result is equal to the final value (namely, zero). When the outcome of the test at step 206 is positive, the routine branches to step 210 to save the value A as the original file that was compressed. This completes the basic processing of the routine.

As can be seen, in a preferred embodiment, the inventive compression routine (and its complementary decompression routine) process the digital file in a temporal or "tim e-based" manner. In particular, the compressed version of the file is a representation (preferably an encoded numeric value) of the amount of time required to process the digital file against a given mathematical function. As the digital file is processed against the given function, a timer is incremented at a given rate. Ideally, the size of the file determines how long the file will take to be processed by the given function (e.g., A = A - 1). When the processing is complete, a value of the timer is then saved as a representation or encoding of the original file. Ideally, the original file is then later reconstructed (i.e., decompressed) from the representation itself by applying an inverse of the given function (in this example, A = A + 1) over the same time period. One of ordinary skill in the art will appreciate that if the same index "n" and the sam e count rate are used for the respective compression and decompression routines, the file A generated by the decompression routine will exactly match the original file input to the compression routine. Thus, the inventive routine provides lossless compression.

In the particular embodiment of Figures 1-2, the index n has a value of 1. In this case, the decrementing function is carried out at each iteration by reducing the value of the bit string by – 1. This function may be readily implemented (at each iteration) by simply complementing the least significant bit (LSB) of the bit string and complementing any higher order bit that has all of its lower significant bits equal to 0. This operation is a conventional down counter. As noted above, during the compression routine, if the result of applying the function is non-zero, the process is repeated for as many iterations as are necessary to decrement the bit

string to the given end value, e.g., zero. As described above, the time taken for this process is the value that represents the "com pressed" version of the original digital file.

The routines illustrated in Figures 1-2 are optimal techniques in which the value of the index "n" does not vary during the compression (or decompression) and is a given value, such as 1. Given the constraints of existing hardware and software, one of ordinary skill in the art will appreciate that the time that may be required to apply the given function (e.g., A = A – 1) to an arbitrary long bit string comprising the file A may be quite lengthy. Thus, according to the present invention, it may be desired to vary the index "n" throughout the given compression routine in the manner now described in Figure 3.

Figure 3 is a flowchart of a preferred compression routine using conventional computational resources. In this example, the routine begins at step 300 by reading an input file A (in effect, a bit string of arbitrary length) that is desired to be compressed. At step 301, a variable i = 1 is set. This variable is used for indexing purposes as will be seen. At step 302, an index $n = n_i$ is set. The routine then continues at step 303, wherein a count $t = t_i$ is set. This value represents a given start value (e.g., a time zero). At step 304, the count is incremented at a given rate. Thus, for example, the given rate may represent the clock speed of the processor being used to compress the file. The routine then enters a given first processing loop as follows.

A test is performed at step 306 to determine whether the file A (or some substantial portion thereof) is greater than a predetermined value, e.g., zero. As indicated in the flowchart, the file A may be partitioned such that all but the least significant bits (the "end of file" or EOF) are processed through this loop. If the outcome of the test at step 306 is positive, the routine continues at step 308 to set a variable B = A. During step 308, the count t_i is paused. At step 310, the routine then applies the given function, namely, $A - n_i$, to A to generate a result. Control then returns to step 306 and the process repeats. During this processing loop, of course, the count continues to increment. When the result of the test at step 306 is negative, e.g., because the value of A (or A|EOF, as the case may be)

is now a negative number (as a result of applying the function in step 310), control then branches out of the first loop to step 312.

At step 312, a test is performed to determine whether the value of A (or A|EOF, as the case may be) is equal to a given end value (in this example, 0). If not, the routine continues at step 314 to save a time value t_1 (in the first iteration) corresponding to the index n_1. Thus, at step 314, the index-time value pair (n_1, t_1) is saved in any convenient location (e.g., in a register, in system memory, on disk, or the like). At step 316, the variable i is then indexed to i + 1. Following step 316, the value A is set equal to the value B. This is step 318. Control then returns back to step 302 and the process repeats again for the new value of the index n. In each next iteration, the count preferably is started anew.

Thus, during a second and any subsequent iteration, other index-time value pair(s) (n_i, t_i) are generated and saved. When, however, the result of the test at step 312 indicates that the value of A is now equal to the given end value, the routine branches to step 320. At this step, the routine saves all of the index-time value pair(s) as the representation of the original data file. Thus, for example, if the given file A went through (i = 3) iterations of the processing loop, the resulting "com pressed" file would be represented as follows (where | is a concatenation operator):

{name | other identifying data | (n_1, t_1) | (n_2, t_2) | (n_3 | t_3) | EOF (optional)} (1)

This completes the compression routine.

Although the values of the index may be selected in any convenient manner, one technique for selecting these values is now described. In a given iteration, the index n is selected by identifying a number of bytes then in the file, dividing the number of bytes by two, and then counting the number of given bit values (e.g., 1's) that are in a given half of the file. Thus, for the first iteration, the index may be generated by dividing by two the number of bytes in the original file and then counting the number of 1's in either half. For the second iteration, the same calculation is then performed on the value B, which is generated in step 318. In an optimal scheme, if the file is odd, then at least one value (e.g., the last iteration) of

the index "n" is 1. If the file is even, then at least one value (e.g., the last iteration) of the index "n" is 2.

In a representative example, the compression routine was used to generate a compressed version of two audio files, file A_1 being a 45Kbyte .wav file and file A_2 being a 75Kbyte .wav file. Using two iterations, with $n_1 = 500$, and $n_2 = 1$, the resulting compressed file were approximately the same size, about 110 - 120 bytes. The time values in each index-time value pair, of course, were different due to the difference in the original size of the files.

One of ordinary skill in the art will appreciate that, regardless of the size of the original file, the resulting compressed version of the file will generally reduce to a relatively small data file that merely encodes the information set forth in equation (1) above. Thus, for example, the inventive compression routine of Figure 3 produced the following results (unless otherwise noted, all files are in bytes):

Sample	Beginning Size	Ending Size
1	72405	102
2	80,302	115
3	2,276,345	123
4	334 Mbytes	117

Figure 4 is a flowchart illustrating how the compressed file representation of equation (1) is decompressed to obtain the original file A. The routine begins at step 400 by retrieving the compressed file representation. At step 401, the routine sets the variable $i = 1$. At step 402, the routine gets a next index n_i. The routine then continues at step 404 to set a count $t_i = - t_i$. Thus, in a first iteration of the decompression routine, t_i is equal to the first time value t_1 that was saved during the compression routine, and n_i is equal to the first index value n_1 used during the compression routine. At step 406, a test is run to determine whether the time increment – t_i is then equal to a given end value, e.g., "0" (or (0|EOF, as the case may be). If not, the routine continues at step 408 to set a value $A = A + n_i$ (which, in the first iteration, is equal to $A = A + n_1$). Control the returns to step 406. When the result of the test at step 406 indicates that the time value has reached the given end value (namely, "0" or 0|EOF), control then branches to step

410. At this step, a test is performed to determine whether there are any more time values "t_i" remaining to be processed. If so, the routine sets $i = i + 1$ at step 412. Control then returns to step 402 and the process repeats with the next index. Thus, in the second iteration, the index – time value pair (n_2,t_2) is processed, during the third iteration, the index – time value pair (n_3,t_3) is processed, and so forth. When there are no more index – time value pairs to process, the outcome of the test at step 410 is negative. Control then branches to step 414 to save the value A as the original file (or the majority portion, if the file is actually A|EOF). This completes the decompression processing.

In an illustrative embodiment, the compression and decompression routines may be asymmetric, with the compression routine taking a longer period of time to execute (in real time) as compared with the decompression function. Primarily, this time differential is due to the fact that some additional temporal overhead is accumulated each time B is set equal to A (step 308) during the compression routine. This operation, as illustrated above, is not required during the decompression routine.

Generalizing, the above-described functionality may be implemented in many different ways, and the present invention is not limited to any particular implementation. One convenient implementation of the invention is in software executable in a processor, namely, as a set of instructions (program code) in a code module resident, for example, in the random access memory of the computer. The clock speed of the processor used to decompress a given file preferably matches the clock speed of the processor used to compress the source file. The invention, however is not limited to having matched processor speeds. In particular, if the speed of a first CPU (used for compression) is "x" while the speed of a second CPU (used for decompression) is "y," then the value of n_i is adjusted during decompression by a constant "k" = n_i (x/y).

In addition, although the various methods described are conveniently implemented in a general purpose computer selectively activated or reconfigured by software, one of ordinary skill in the art would also recognize that such methods may be carried out in hardware, in firmware, or in more specialized apparatus constructed to perform the required method steps. A preferred implementation is a

dedicated device, such as a integrated circuit chip, that includes its own on-board processor, memory and system clock. In such case, the compression and decompression routines are stored in firmware or the like. Figure 5 illustrates a simplified configuration of one such device. In this example, the chip 500 includes a processor 502, system memory 504, firmware 506 (comprising the compression/decompression routines), and a system clock 508. The device is connectable to a computer motherboard, for example, via the link 510. Link 510, for example, may be an RS232, USB, or the like. When a given file A is desired to be compressed, the file is provided to the chip 500 in the usual manner. The resulting compressed file representation is then returned.

The inventive routines may also be implemented in other known hardware such as a digital signal processor (DSP), a field programmable gate array (FPGA), an application specific integrated circuit (ASIC) chip, a microcontroller, in hardwired logic, or the like.

The codec of the present invention offers virtually unlimited lossless, content-insensitive compression. As a consequence, the applications for the codec are quite diverse. The following are several representative applications, although this description should not be taken to limit the scope of the present invention.

Even in its software-only form, the codec may be used as a substitute for existing archival algorithms. Once the codec is ported from software source code to a dedicated integrated circuit, it can be used to retrofit existing storage media with unprecedented usable capacity and allow enormous amounts of data to be transmitted across existing telecommunications and data communications infrastructures. The hardware codec may also be paired with custom-tailored operating systems that manage memory structures, with custom hooks into codec-driven storage devices and codec-enabled networking hardware.

Software Applications

The following are areas in which the codec may be employed in software under existing operating systems.

- Electronic Software Distribution (ESD): HTTP- and FTP-based distribution of software has exploded with the rise of e-commerce. Using the codec for file compression (as a product is only packaged for ESD once, the time needed for

file compression would not be a problem) and wrapping up a freely-distributable decoder in the ESD package, any software company could sell their products on-line, regardless of file size. In this application, the processor-speed compatibility is achieved by (a) posting different compressed file versions to match, say, 10 or 15 available clock speeds, or, in later-generation PCs, (b) querying the embedded processor serial number to retrieve the correct download without user intervention.

- Web Browser Acceleration: With a codec-enabled web browser (through a Netscape plug-in or ActiveX control) and a web site posting compressed HTML pages, JPEGs, and GIFs, bringing up a page would be hundreds of times faster. This application would work for static pages, applets, static frames, images, and prerecorded streaming audio and video. Moreover, because the codec would not alter fundamental Web content, it could be offered as a third-party add-in to existing web servers and supplied as a free add-in for existing web browsers.

Hardware Applications

Once ported to integrated circuit or other such hardware device with integrated processor capable of real-time compression, the codec provides additional improvements in performance. The chip has applications in both the data communications vector and data storage vector.

Data Communications Vector

- WAN Acceleration Appliance: One application of the Codec is in a hardware "black box" (chip and custom firmware) on both ends of a wide-area data transmission. Data packets are transmitted in raw form over a local Gigabit Ethernet or ATM LAN, then compressed right before entering a slow-speed transmission medium (i.e. between the last router and its CSU/DSU). At the other end, an identical box decodes the data stream. Using the present invention, full-motion streaming video and audio would not only be possible but commonplace. The device could be used for both Intranet (that is, connecting multiple sites within a company) and Internet connectivity – the latter requiring that one's IS P use the device. The codec may also be used to implement an Acceleration Appliance, an end-node version with a lower-end processor to

handle a certain volume of traffic in real-time; and an Analog Acceleration Appliance, targeted at asynchronous Internet connections to local POPs.

- LAN/WAN Acceleration Chip: The integration of the hardware codec directly into routers, switches, and network interface cards would be quite useful. Soldering the chip to hardware and codec-enabling firmware would (a) reduce overall costs of implementing the codec, (b) enable "codec to the desktop" through integration onto NICs and within switch/router backplanes; and (c) simplify internetworking architectures by reducing the number of devices involved. The IC would have such a small physical footprint that it could easily be implemented in a wide range of Ethernet-compatible interfaces, from laptop PC Cards to 802.11 wireless ports to 1000Base-T NICs.
- RS-232/RS-422 Acceleration Module: While data transmission using serial protocols will always have physical speed limits, placing a host-powered codec "dongle" betw een the serial port and the cable medium at both ends would have a dramatic impact on data collection and instrumentation monitoring in manufacturing environments, eliminating all bottlenecks due to old-school transmission media.
- Satellite Communications Accelerator: As with the similar products above, this would codec-enable both orbiting reconnaissance, cable, and telecommunications satellites and out-of-orbit research satellites. The time needed to transmit celestial images, sports simulcasts, and other photos would be negligible; when used on earthbound wireless media, mobile phone fidelity would be perfect due to a higher sampling frequency and wireless CDPD data connections, currently restricted to 19.2 Kbps, could come up to current leased-line speeds.

Data Storage Vector

- DASD Interface: The DASD Interface would be an IC-based traffic accelerator for the DASD (Direct Access Storage Device, or hard drive) communication channel.
- DASD Compression: Codec-enabled motherboards would allow OS storage drivers to pass all data through the codec before it hits the DASD, and decode the data automatically on the way back.

- ROM/NVRAM Retrofit: With the addition of a hardware codec, the existing chips in cell phones, Palm OS handhelds, digital watches – any device currently equipped with non-volatile memory of any kind – would have a effectively limitless off-line storage capacity and a real-time data access capacity limited only by their working RAM.
- ROM Compression: The hardware codec would also make practical an entire new generation of portable devices built from the ground up with huge storage capacities: handheld libraries, wallet-size medical IDs with an copy of your extended family's m edical history and your DNA fingerprint on-board, portable audio players with a true-fidelity copy of a large number of CDs in current release, and feather-light laptops booting an operating system such as Windows NT from ROM.
- NVRAM Storage Devices: By combining the size, speed and portability of existing flash media such as the Sony Memory Stick with codec-enabled motherboards, an vastly improved device is created. Zip disks and other "high-capacity" removable storage devices would not be required.
- Digital Tape Compression: codec compression for long-term archiving is another application. By embedding the codec IC into DLT or DDS tape drive hardware, a business can keep decade-long digital histories on a single tape.
- Medical Applications: The following are exemplary: (a) enabling the real-time transmission of MRI and X-ray images, the most complex loss-sensitive digital images in the world, currently compressed using the CCITT algorithms, over any connection; (b) medical monitoring equipment would be able to keep a complete signal history at a very high sample rate, rather than having to dump data histories to printers or overwrite them; and (c) replicated databases of patient information could be kept at every hospital in the world, rather than being haphazardly transmitted after emergencies or transfers.
- New Operating Systems: Operating systems developers may write code that uses the codec's on-board logic to m ultiply and manipulate system memory structures with an efficiency never before possible.

Having thus described my invention, what I claim as new and desire to secure by Letters Patent is set forth in the following claims.

CLAIMS

1. A method for processing a digital file comprising a bit string, comprising the steps of:

initiating a count that increments at a given rate;

as the count increments, iteratively decrementing the bit string, by a given value, until a given end value is reached; and

upon reaching the given end value, saving a then-current count increment as a representation of the digital file.

2. The method as described in Claim 1 wherein the count is a timer.

3. The method as described in Claim 1 wherein the given binary value is 1.

4. The method as described in Claim 1 wherein the end value is 0.

5. The method as described in Claim 1 further including the step of reconstructing the file by:

setting the representation to a given count value and incrementing the given count value at the given rate;

as the given count value increments, iteratively incrementing a given number, by the binary value, until a given end value is reached; and

upon reaching the given end value, saving a then-current value of the given number as the file.

6. A method of compressing a file A, comprising the steps of:

(a) getting a next index n_i, where $i = 1, 2, \ldots$

(b) initiating a count t_i, wherein $I = 1, 2, \ldots$

(c) as the count t_i increments:

(1) determining if A is greater than 0;

(2) if so, setting $B = A$;

(3) calculating $A = A - n_i$; and

(4) repeating step (c)(1) until A is greater than 0;

(d) when A is greater than 0, determining if A is equal to a given value;

(e) if not, setting A = B;

(f) saving an index, time value pair (n_i, t_i);

(g) setting i = i + 1;

(h) repeating steps (a)-(f); and

(i) if A is equal to the given value, concatenating the index, time value pairs as a compressed version of the file A.

7. The method as described in Claim 6 wherein the given value is 0.

8. The method as described in Claim 6 wherein the given value is (0|EOF).

9. The method as described in Claim 6 wherein a given value of the index is calculated by:

dividing a number of bytes of the file A by a given number;

determining a number of bits in a portion of the file having a given value;

setting the index equal to the number of bits.

10. The method as described in Claim 6 wherein a given value of the index n_i is equal to 1.

11. The method as described in Claim 6 wherein a given value of the index n_i is equal to 2.

12. A method for decompressing a file represented by a concatenation of index, time value pairs, comprising the steps of:

(a) getting a next index n_i, where i = 1, 2, ...

(b) initiating a count t_i, wherein i = 1, 2, ...

(c) as the count t_i increments from a value - t_i:

(1) determining whether t_i is equal to a given value;

(2) if not, calculating $A = A - n_i$; and

(3) repeating step (c)(1) until t_i is equal to the given value;

(d) determining whether there are any count values remaining to be processed;

(e) if so, setting $i = i + 1$;

(f) repeating steps (a)-(e);

(g) if there are no count values remaining to be processed, saving A as the file.

13. The method as described in Claim 12 wherein the given value is 0.

14. The method as described in Claim 12 wherein the given value is (0|EOF).

15. A computer program product in a computer readable medium for compressing a file represented as a bit string, comprising:

a set of instructions executable in a processor for performing the following compression method steps:

initiating a count that increments at a given rate;

as the count increments, iteratively decrementing the bit string, by a given value, until a given end value is reached; and

upon reaching the given end value, saving a then-current count increment as a representation of the file.

16. The computer program product as described in Claim 15 further including:

a set of instructions executable in a processor for performing the following decompression method steps:

setting the representation to a given count value and incrementing the given count value at the given rate;

as the given count value increments, iteratively incrementing a given number, by the binary value, until a given end value is reached; and

upon reaching the given end value, saving a then-current value of the given number as the file.

17. The computer program product as described in Claim 16 wherein the processor used to perform the decompression method operates at a clock speed equal to the clock speed of the processor used to perform the compression method.

18. An integrated circuit codec, comprising:

a processor;

a clock operating at a given clock rate; and

code executable by the processor at the given clock rate to compress a file represented as a bit string, by:

initiating a count that increments at the given clock rate;

as the count increments, iteratively decrementing the bit string, by a given value, until a given end value is reached; and

upon reaching the given end value, saving a then-current count increment as a representation of the file.

19. The integrated circuit as described in Claim 18 further including:

code executable by the processor at the given clock rate to decompress the representation to regenerate the file, by:

setting the representation to a given count value and incrementing the given count value at the given rate;

as the given count value increments, iteratively incrementing a given number, by the binary value, until a given end value is reached; and

upon reaching the given end value, saving a then-current value of the given number as the file.

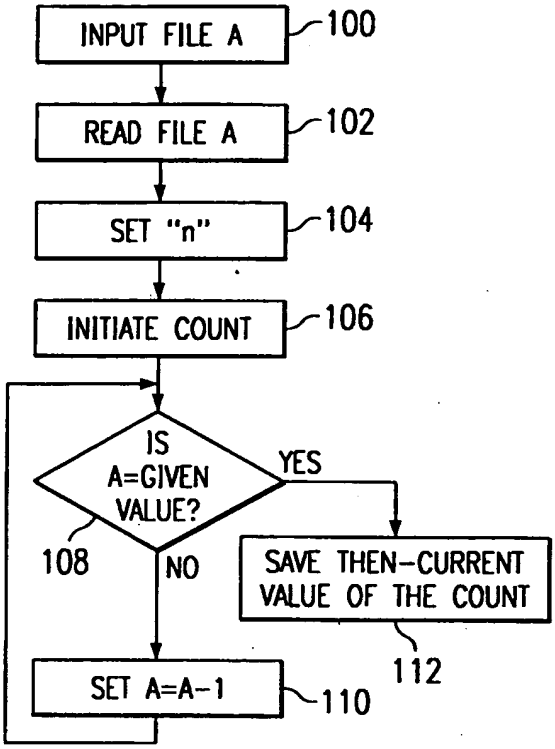


FIG. 1

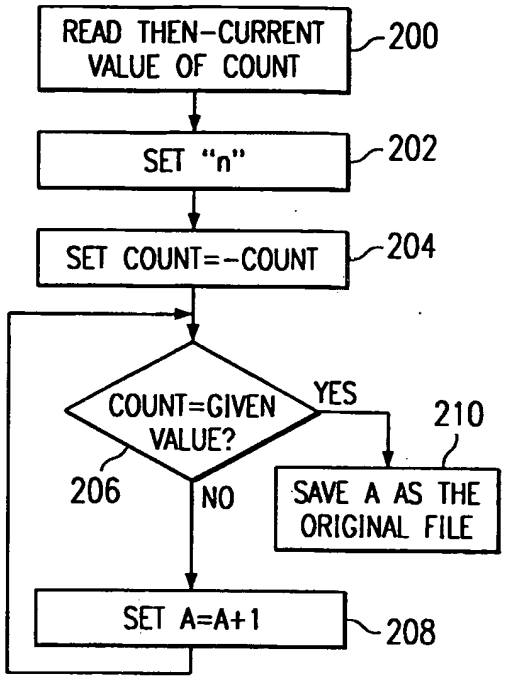


FIG. 2

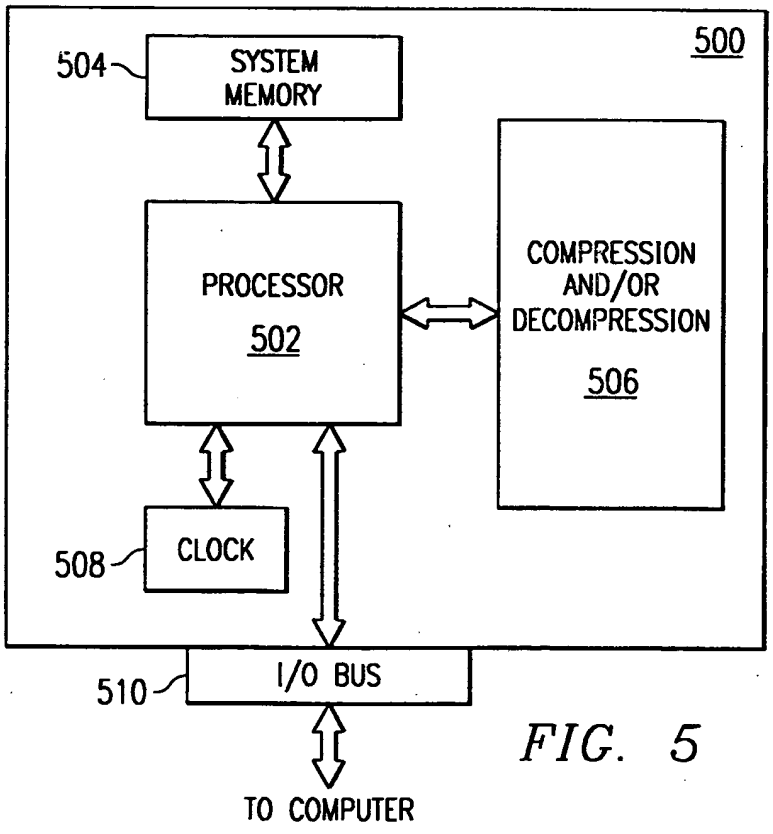


FIG. 5

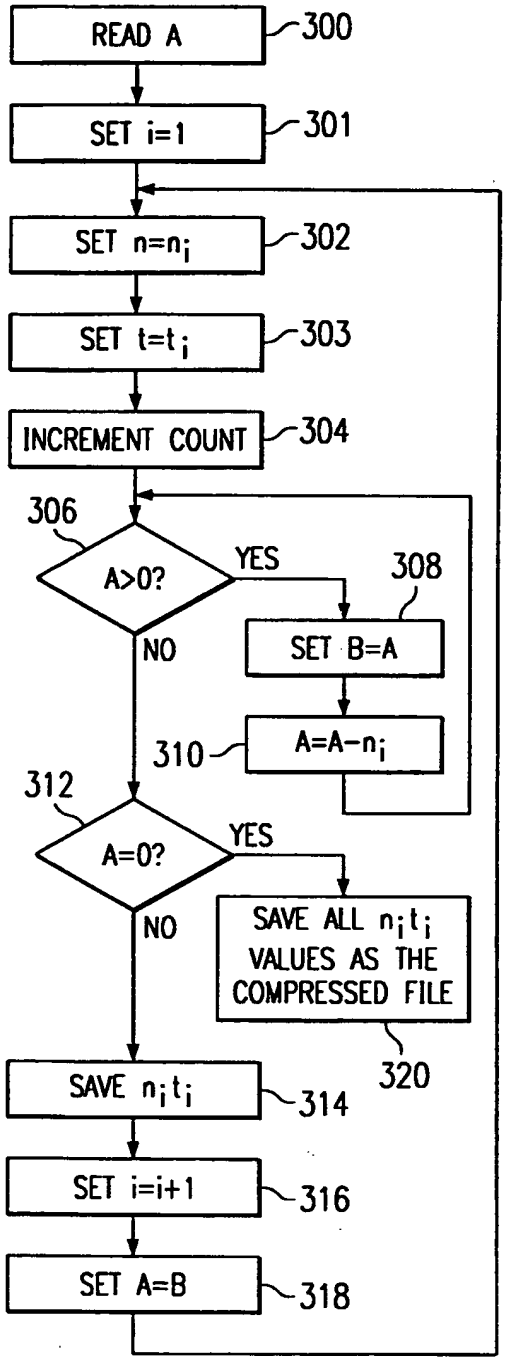

READ A
300
SET i=1
301
SET n=n_i
302
SET t=t_i
303
INCREMENT COUNT
304
306
A>0?
YES
308
SET B=A
NO
310
A=A-n_i
312
A=0?
YES
SAVE ALL $n_i t_i$ VALUES AS THE COMPRESSED FILE
320
NO
SAVE $n_i t_i$
314
SET i=i+1
316
SET A=B
318

FIG. 3

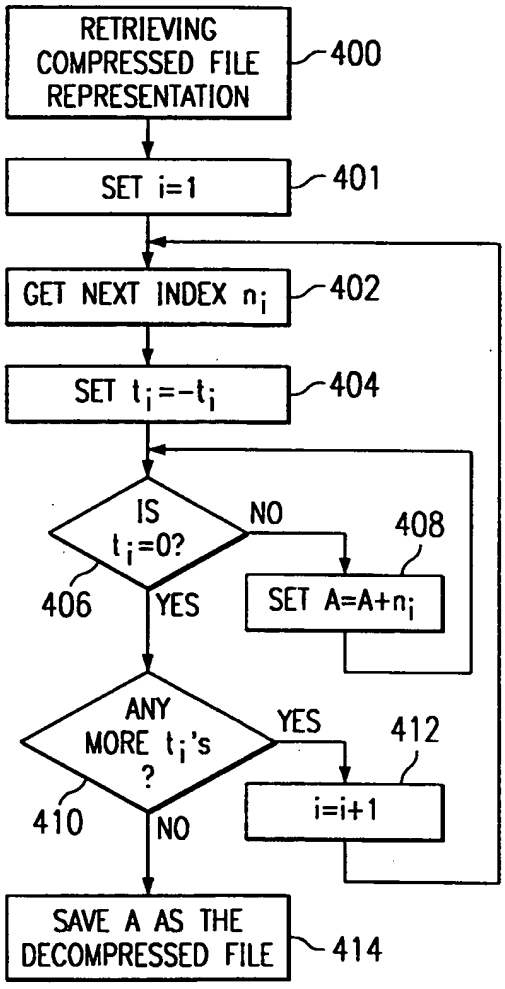

RETRIEVING COMPRESSED FILE REPRESENTATION
400
SET i=1
401
GET NEXT INDEX n_i
402
SET t_i=-t_i
404
IS t_i=0?
NO
408
406
YES
SET A=A+n_i
ANY MORE t_i's ?
YES
412
410
NO
i=i+1
SAVE A AS THE DECOMPRESSED FILE
414

FIG. 4